A Template-Based Mechanism for Dynamic Service Composition Based on Context Prediction in Ubicomp Applications

Ángel Jiménez Molina, Hyung-Min Koo, In-Young Ko
Information and Communications University, 119 Munji-Ro, Yuseong-Gu, Daejeon, Korea
{ajimenez, hyungminkoo, iko}@icu.ac.kr

Abstract

In ubiquitous computing environments, an application describes a task that needs to be performed to satisfy a high-level user's goal. When an application is executed it is important to understand the context of the user and his/her surrounding environment, in order to dynamically instantiate a task by using available services. This task-instantiation process faces the problem of complexity in choosing appropriate services for a task. In order to improve the performance of dynamically choosing and binding services, in this paper we propose a template-based mechanism to dynamically select and compose services according to the context information predicted in advance. This mechanism ensures continuous service provision, service conflict prediction and resolution, and dynamic reconfiguration of services. To make use of predicted context, our mechanism provides the following capabilities: (1) task-based context prediction, (2) template-based task execution and (3) pattern-based service composition. We developed a task-oriented architecture to support these capabilities. Our dynamic service composition mechanism contributes to improve the reusability, flexibility and dynamism of ubiquitous computing applications.

Keywords: *Ubiquitous computing, context prediction, context awareness, task-oriented computing, architecture-driven systems.*

1. Introduction

Ubiquitous computing (ubicomp) is a forward state of distributed systems (DS) and mobile computing (MC). Some of the supporting technologies (embedded devices, components services, communications capabilities and others) in ubicomp correspond to those in DS and MC (Satyanarayanan, 2001) [13]. The difference is that ubicomp is user-centric. That is, such ubicomp applications (UCAs) must satisfy users' goals, which are usually represented in tasks [18]. In order to satisfy tasks, UCAs need to gather users' goals and context information [11] through embedded devices and others.

Context information corresponds to the user's state and his/her space [14]. For instance, physical location, emotional state, physiological state, daily recurrent activities, users' profile, or the room temperature, light and humidity, network bandwidth, etc. When an UCA is executed, it is important to understand and interpret the context information, and based on it, execute adequate services to satisfy high-level users' tasks. Services that meet tasks may come from multiple heterogeneous computing resources, which are usually connected by the means of a middleware infrastructure. This infrastructure dynamically instantiates tasks into available services, which need to be conveniently configured, initialized and composed [1], [4], [17], [15].

The task-instantiation process must occurs with the minimum delay from the UCA identifies the task until its satisfaction. In this sense, our work strives to incorporate proactivity into the UCA. In a proactive system, the state at time t depends on the current, past and predicted future states [12]. As was explained before, the state of the user and his/her space corresponds to the context information. Thus, our idea is to provide to the UCA not only with the current context information, but also with predicted context [cf. 12]. In order to enable the UCA to proactively deliver services, this predicted context need to be of high-level. In Ubicomp, high-level context is represented in tasks.

A proactive as well as a reactive task-instantiation process faces the problem of complexity in selecting appropriate services for a task. This complexity can be reduced by improving the performance of dynamically selecting and binding services. In this sense, we propose a template-based mechanism to dynamically select and compose services according to the context information predicted in advance. This mechanism ensures continuous service provision, service conflict

prediction and resolution, and dynamic reconfiguration of services. Our dynamic service composition mechanism provides three major capabilities: (1) task-based context prediction –making use of the work of Mayrhofer et al. [12]-, (2) template-based task execution and (3) pattern-based service composition. The organization of those capabilities results in a task-oriented architecture.

Our solution contributes to improve the reusability, flexibility and dynamism of UCAs. In fact, templates and patterns can be reused for others UCAs, which can be made by composing patterns. That is, patterns can be composed dynamically to meet dynamically-changed user's tasks. Current ubicomp frameworks do not provide such template and patterns reusability.

The rest of the paper is organized as follow. In section 2 we introduce the related work in ubicomp and context prediction. Section 3 provides a brief explanation of task-oriented applications execution and the design of our task-oriented architecture. Section 4 provides the dynamic template-based service composition mechanism. In Section 5 we describe a sample scenario for our service composition mechanism. In section 6 we evaluate our mechanism. Finally, we draw a conclusion and explain future work in section 7.

2. Related Work

Ubicomp is an active research issue. In fact, several abstract frameworks have been proposed to develop UCAs that use context information. However, in relation to our work, can be mentioned those that use historical data. In that category we can find examples like CASS [7], CoBra [6], Context Toolkit [2], CORTEX [4], Gaia [8], SOCAM [9], etc. However, none of them provide a proactive way of selecting and binding services. The Aura project [18] has works related to computing resource prediction during run-time. The goal of this prediction is to maximize users' utility allocating resources to UCAs based on an instantaneous evaluation of configuration. However this work does not consider high level context prediction, which corresponds to the task desired by the user, inferred from the user's state and his/her space.

There exist some works related with predicting high-level context from user behavior, which, for instance, have been addresses by Mayrhofer et al. in [12]. In fact, in this work, user behavior in several domains can be gathered and analyzed in order to predict future user context from historical data, delivering proactive services to the user. This prediction depends on the current and future state of the environment and UCA context, which is captured for suitable sensors.

According to [12], one of the most important constraints in future context is that its capture and prediction have to be during run-time, in order to ensure that running applications do not stop and continuously provide services to the user. On the other hand, the system also might provide prediction of the available resources, such as network bandwidth or memory space [18]. Unfortunately, usually there are not enough log data for analyzing it offline, such as in knowledge discovery in databases approach. Therefore, our approach is based on providing future context prediction during run-time.

An architecture for context prediction has been introduced in the work of Mayrhofer et al., which can be implemented making use of specific type of sensors, classification algorithms and forecast techniques. In order to achieve their goals, the authors propose the use of classification algorithms, which can determine which action is the closest to the current situation in order to proactively prepare a future task execution.

This architecture constitutes an application framework, which defines the following logical steps for deriving future context from low-level raw sensor data: sensor data acquisition, classification, labeling and prediction (Fig. 1). The rationale to select the Mayrhofer's work is that based on raw sensor data (low-level context) it is possible to infer high-level context, which enable the UCAs to perform tasks in advance.

In short, in order to forecast context to improve the task plan generation and deliver proactive services to the user during run-time, it is possible to process and classify gathered raw sensor data of a current state, enabling the prediction of future context based on this historical data.

2.1 Context Prediction Model

This section introduces the context prediction model to be used as part of our dynamic template-based service composition mechanism. As was stated in the second section, this model is based on the work of Mayrhofer et al. in [12]; however, in this paper is provided our own interpretation of the model.

The first definition for the context prediction model corresponds to "internal context" of the UCA at time t, which is established, based on the sensor data. Internal context depends on the current context, the last context and the input value at time t-1. The model output at time t depends on this internal context and on predicted future contexts. Future contexts are predicted

recursively making use of suitable algorithms, such as clustering, neural networks, time series techniques, etc. [12]. The complete process considers the following steps:

i. *Sensor data acquisition:* raw sensor data gathered at time t, defines a sensor vector s in the sensor space S, with $(s_1 \times s_2 \times \ldots..x\ s_L)_t \in S_1 \times S_2 \times \ldots..x\ S_L$, which dimension is adjusted to the nature of the sensor measurement (brightness, audio, temperature, etc.)

ii. *Feature extraction:* the sensor vector s is transformed into the feature space F, contracting or expanding it in order to better interpretation and computation management. S → F. The new vector f in F is called feature vector, with $(f_1 \times f_2 \times \ldots..x\ f_n)_t \in F_1 \times F_2 \times \ldots..x\ F_n$. For instance the sensor vector can be scaled to the interval [0,1].

iii. *Classification:* the context predictor finds out common patterns in the feature space, defining clusters. In this sense, the feature vector in F is assigned to one cluster c, or multiple clusters with different degree of membership, in the space C, with $(c_1 \times c_2 \times \ldots..x\ c_m)_t \in C_1 \times C_2 \times \ldots..x\ C_m$. The classes $\{c_i\}$ constitute the detected context in the space C.

iv. *Labeling:* in this step each class is assigned to predetermined strings in N that denote descriptive names. These descriptive names are defined according to the current space features. In this sense, $C_1 \times C_2 \times \ldots..x\ C_m \rightarrow N$. A context label $n_t \in N$ describes the current context at time t.

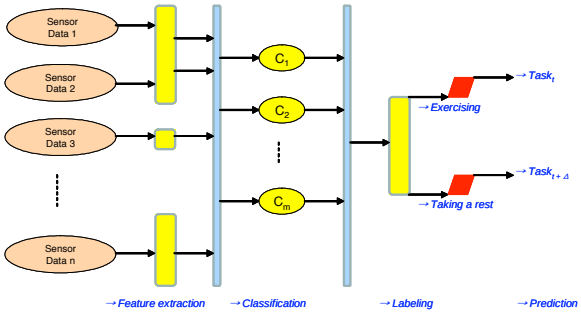

Fig. 1 Context Prediction Model

v. *Predicition:* It is defined the future class vector $(c_1 \times c_2 \times \ldots..x\ c_m)_s = p((c_1 \times c_2 \times \ldots..x\ c_m)_t, t, s)$, whit $(s > t)$ and p an iterative process. The results are the future classes in s, based on time t, which is the predicted context.

3. Tasks-Oriented Architecture

Task-driven computing enables the UCA to access explicit computing tasks and task context, automatic service configuration and composition, manage computing resources constraints and dynamic services. In this sense, the benefits to users are that they can accomplish high-level tasks with transparency of low-level configuration activities (manual configuration of single devices such as desktop computers, laptops, PDAs, or manual transference of users; state among those devices, or mapping of their tasks by themselves)

In this paper, a task is defined as a set of *actions* organized in a sequential or parallel way. An action is a high-level function defined according to the user's perspective. In this sense, a task attempts to formally and explicitly represent high-level user's goals, such as preparing a presentation, taking a rest, editing a document in the airport, meeting scheduling, etc.

In real world, users require to move from one to another task or to continue performing a specific one. In short, it is task transition and task continuity respectively. In fact, given an interruption of a task, the system has to be able to realize when the user attempts to retake it, providing continuity in the same or another space. In addition, task transition is verified when users require to change from one to another task.

The organization of these actions constitutes a task plan, which involves a generation effort to the UCA that can be reduced. In fact, making use of high-level predicted context, can be determined a future action of other task, that switch with the current action in a given task. Actually, can be determined several different tasks that match with the current action. Therefore, in order to reduce the efforts to generate the task plan, there exists the need of a mechanism to decide between the set of predicted future tasks. However, it is not in the focus of this paper.

As is noted in this paper, there is a benefit utilizing the capability of context prediction in order to dynamically compose services in advance. In addition, this goal is pursued through two more capabilities provided by this paper: template-based task execution and pattern-based service composition. These capabilities are organized in a task-based architecture.

The fundamental definitions provided in this paper to design the task-oriented architecture are: *application template* and *service composition pattern*. In fact:

i. ***Application template***: a determined and

descriptive way to organize actions in a suitable sequence (sequential or parallel)

ii. ***Service composition pattern***: a configuration or set of high-level abstract services or functions that are to perform actions.

In order to provide the required task-based architecture, let's consider the following overall structure composed of its core elements. Main elements of our architecture are as follow (see figure 2):

i. ***Context Manager****:* it receives low-level raw sensor data and performs high-level task context prediction, making use of the *context predictor element*. In addition, given the predicted task, an application template is selected from the *repository* through the *application template broker*.

ii. ***Reconfiguration Manager***: it deploys *service composition patterns* and makes a consolidated service composition pattern (for service conflict prediction and resolution)

iii. ***Service Composition Pattern Broker****:* it searches appropriate service composition patterns to support actions in application template.

iv. ***Distributed repositories****:* they store application templates and service composition patterns in distributed environment. In addition, it provides a way to search appropriate application templates and service composition patterns from distributed repositories.

v. ***Application Composition Tool****:* this tool supports developers for developing, registering into the repositories and simulating application templates and service composition patterns.

The context manager acts given a current task. Indeed, its components are the context predictor and the application template broker. The context predictor is in charge of determining proactively the next task to be executed, based on the raw sensor data gathered by the devices, and the historical information keeps into the *context base repository*. The context base repository holds high-level and low-level information. Regarding to high-level information are the tasks, actions and required quality attributes of them. This information must be determined in advance and is part of design work. Indeed, this repository stores the history of switched tasks. Regarding to low-level information, it is stored current and historical raw sensor data. A deep description of the context base repository can be done through an ontology description model, but it is not in the scope of this paper. The application template broker gets the predicted task from the *application template repository*.

The reconfiguration manager contains the service composition pattern broker, which gets the service composition patterns from the *service composition pattern repository*.

The requesting of services is done by the *service discovery*, which interacts with the *service repository*. There are several protocols that can be used in our task-based architecture to accomplish the service discovery process, such as UPnP, Jini, Salutation, Service Location Protocol, Group-based Service Discovery, etc. In all these protocols, service providers make an advertisement of their services description and attributes, following broadcast, unicast or multicast strategies. For instance, in Jini each service provider finds out a lookup service to inform its service object and service attribute. Then, a client requests a service downloading the mobile code of the service object. This service object is used by the client to invoke the actual service from the service provider [19]. In this example the lookup service is like a directory repository that in ubicomp environments is usually distributed.

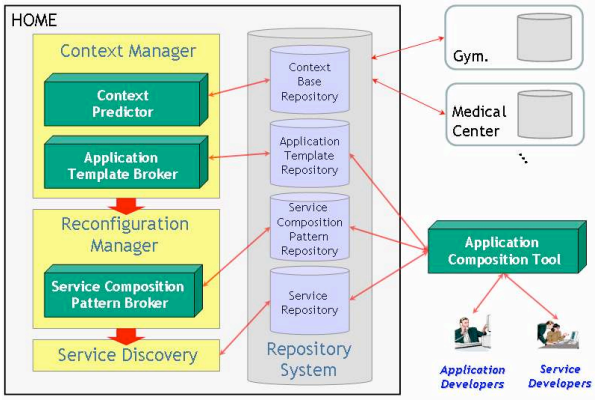

Fig. 2 Overall Task-Oriented Architecture.

4. Dynamic Template-Based Service Composition Mechanism

In this section, we describe our proposed service composition mechanism, to improve the performance of dynamically selecting and binding available services according to the context information predicted in advance, reducing the complexity of this process. This

mechanism makes use of our introduced application template and service composition pattern definitions.

The mechanism has as input low-level raw sensor data that generate the current task context (state). Consequently, it predicts during run-time the context determining next tasks that have to be performed, enabling to UCA proactively configure, initialize and compose accessible services in advance. In this sense, proactivity ensures that the task-instantiation process is verified with the minimum delay from the UCA identifies the task until the delivery of services to the user.

As is shown in the figure 3, the mechanism introduced in this paper, begins with high-level user's goals represented in tasks. Given a task, the mechanism considers a medium-level represented in actions. For each action are selected a set of service composition patterns, and finally several actual services to satisfy the user's goal and execute the task.

Therefore, our dynamic template-based service composition mechanism considers applying the following steps:

i. The context predictor receives low-level raw sensor data gathered from physical sensors. Indeed the context predictor selects historical raw sensor data and high-level information from the context base repository.

ii. The context predictor makes use of the task context prediction model introduced in the section two and forecast the next tasks.

iii. Given the predicted task[1], the context predictor communicates the task features to the application template broker, element that is in charge of gather in advance an adequate predetermined task from the application template repository. In advance means before the task is actually required to be executed by the UCA.

iv. Once selected the adequate task that match well with the predicted one's features, the application template broker gets the set of actions that conform the task and that will be necessary to map to service composition patterns.

v. For each action, the service composition pattern broker must search service composition patterns into the service composition pattern repository.

vi. Given the service composition pattern, the service discovery requests actual services from the service repository or other distributed repositories. The searching trials into the service repository are delimited by the required functionality contained in each given service composition pattern.

vii. The actual services are bounded and executed to perform the service composition pattern, and consequently perform task actions.

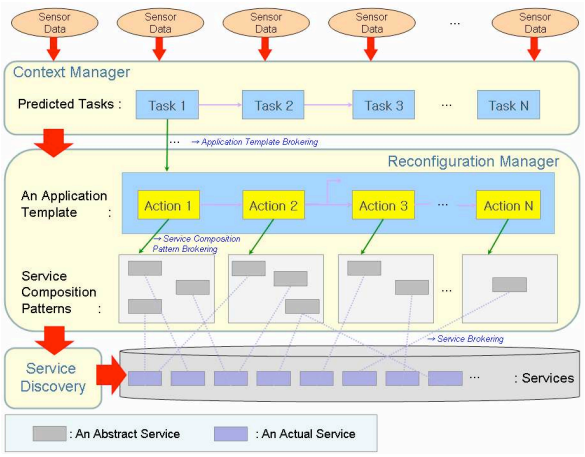

Fig. 3 Dynamic Template-Based Service Composition Mechanism.

Our proposed mechanism can be described in pseudocode as depicted in figure 4.

5. Sample Scenario

In this section, we describe a simple scenario of using the service composition mechanism. This scenario shows how the UCA makes use of predicted context to compose in advance the services that are required for a future task. The involved tasks are: "exercising" task and "taking a rest" task in a home space. We assume that there is one user at home, and that user's hobby is exercising.

i. John comes to home from his job.

ii. John changes his suit to training cloths, and then context manager finds out that John wants to exercise and select the "exercising" task.

iii. The application template for exercising is selected and prepared for execution.

iv. John goes to the area where the exercising

[1] In the eventuality of multiple predicted tasks, it is necessary to perform a conflict resolution. This resolution can be done under several strategies, such as working with priorities of tasks or performing negotiations between them. However it is not in the scope of this paper and it is proposed as future work of our research.

machines are.

v. The application template for exercising is activated by the reconfiguration manager.

vi. Actions (e.g. "making the room cooler", "making the exercising machines available") and services (e.g. "open the windows", "turn on the running machine") are executed.

vii. During the exercising, the context manager collects sensor data (e.g. temperature of John's body and the room, passed time of exercising, burnt calories of user).

viii. The context manager predicts that the user will take a rest after exercising and select "taking a rest" task. Then the application template for "taking a rest" is selected.

ix. When John finishes his exercises, the application template for taking a rest is activated and actions (e.g. "making room darker", "making room quiet") and services (e.g. "make lights darker", "make AV devices quiet") are executed (See figure 5)

```
//Given a current task
Gather_Sensor_Data () {
        s := sensor_vector;
}
Gather_context_repository_information ()
        hi := historical_information;
}
Context_prediction (s, hi) {
        Feature_extraction () {
                f := feature vector;
        }
        Classification (f) {
                c_i := class_i; //Detected user context
        }
        Labeling (c_i) {
        n_t := context_t; //Predefined context names
        }
        Prediction (n_t);
}
Get_task_application_template_repository
(predicted_task); //Application template broker
Get_actions (task); //Application template broker
For each action_i {
        Get_service_composition_patterns_repository
        (action_i); //Service composition pattern broker
        For each service_composition_patterns_j {
                Get_actual_services
                (service_composition_patterns_j);
                //Service discovery
        }
}
Execute_actual_services (actual_service_k)
```

Fig. 4 Pseudocode of the Dynamic Template-Based Service Composition Mechanism.

Application templates and service composition patterns are described making use of ACME architecture description language, as preliminary depicted in figure 6.

6. Evaluation

Main feature of our mechanism includes the high-level abstract representation of actual services into service composition patterns, which reduces the necessary searching trials into the set of primitive services. Therefore, this mechanism reduces the complexity of selecting suitable services for a task, and improves the required process. As this abstract representation is linked with the user's goals through the reusable application templates, which are not present in other popular ubicomp frameworks such as Aura [18], it inherits the required functionality from a high-level user's perspective. Therefore, our task-oriented architecture and template-based mechanism constitute a topological optimization to enable the dynamic service composition process to reduce its inherent complexity. The Following features are implied from our dynamic template-based service composition:

i. ***High-level abstract service composition****:* services can be composed based on user-perspectives, rather than system-perspectives.

ii. ***Dynamic reconfiguration of services****:* services and their composition can be dynamically reconfigurable based on changes of available services and environment conditions.

iii. ***Continuous Service Provision:*** predicted context enables the UCA to prepare the execution of tasks in advance. It reduces interruptions to service provision, which is performed without a hitch.

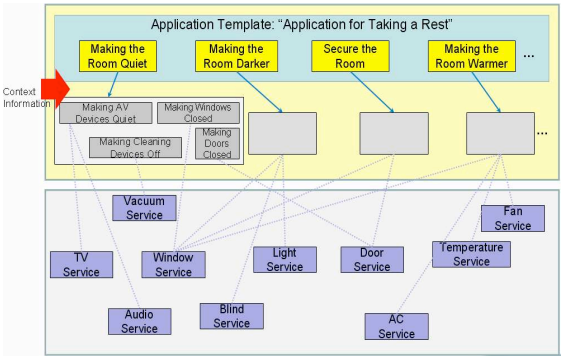

Fig. 5 Taking a Rest Task.

iv. ***Service conflict prediction and resolution:*** conflicts can be detected and resolved by using high-level abstraction mechanism, before executions of actual services. It enables the UCA to improve the use of its limited resources.

We are currently working in the implementation of a prototype of our task-oriented architecture to perform experiments in our test-bed [20]. We will evaluate the performance of our service composition mechanism through two specific metrics: number of service provision interruptions and the required searching trials of actual services in long term execution. By the means of the first metric, it is possible to perform a binary evaluation of the service provision continuity. The second metric is a way to evaluate the reduction of complexity in searching suitable services to execute a given task, and in switching them.

7. Conclusion

In this paper we provide a task-oriented architecture and a template-based mechanism for dynamic service composition in UCAs based on the predicted context. We have introduced two key definitions considered in our architecture and mechanism: application template and service composition pattern. By the means of the template-based mechanism, the UCA makes a coordinated transition in advance between different tasks based on predicted tasks, reducing the complexity of the service composition.

Our solution contributes providing an improvement of UCAs' reusability. In fact, an application template constitutes a reusable structure for a specific task, and the service composition patterns can be reused for other UCAs. In addition, there is an improvement of flexibility that can be incorporated during design time.

```
System making_the_room_quiet ={

Component Making_AV_Devices_quiet= {

        Component AV_Services = {
                ports send;
                operations{AV_service.setVolmueLevel(2);}
                properties {rdfsource : string = http://as.icu.ac.kr/services#av_services}}
                Connector AudioRequest = { Role { request; response } }
        Attachments {
                                .....
        }
}

Component Making_cleaning_devices_off ={
        operations{cleaningDevices.turnOff()}
        properties {patternSource : string = http://as.icu.ac.kr/patterns#making_doors_cloased}
}

Component  Making_doors_closed ={

        operations{doors.close()}
        properties {patternSource : string = "http://as.icu.ac.kr/patterns#making_doors_cloased"}
}

Component  Making_windows_closed ={
        operations{windows.close()}
        properties {patternSource : string = "http://as.icu.ac.kr/patterns#making_windows_closed"}
}

}
```

Fig. 6 An example of Service Composition Patterns Description Written in ACME

In fact, UCAs can be made by composing patterns, and patterns can be used for many UCAs, rather than considering composition of actual services.

Also our solution contributes with an improvement of dynamism. Patterns can be composed dynamically to meet dynamically-changed user's goals and to deal with dynamically-changed space conditions, based on context information. Patterns provide units for dynamic reconfiguration of UCAs during run-time, rather than reconfiguration of real services. In addition, our task-oriented architecture can provide user-perspective services by using high-level patterns and services, rather than system-perspectives. In this sense, there is a separation of users' concern from actual services.

Our future work considers the performance evaluation of the proposed service composition mechanism through two specific metrics: number of service provision interruptions and the required searching trials of actual services in long term execution.

We are currently working on the problem of service conflict prediction and resolution in an abstract level. For instance, the application template "making the room quiet" will require the service composition pattern "making windows closed". However, an application template such "making the room cooler", will require the service composition pattern "making windows opened". In this sense the conflict should be predicted and resolved through some mechanism, such as providing alternative services, working with priorities of services or performing negotiations between services.

An interesting future work is related with executing an off line context prediction. In this case it will be necessary to design a way to detect common patterns into tasks. These patterns can be deduced from the historical data making use of clustering algorithms that create groups of related information. Patterns can be used in the future for a better reaction of UCAs. Also patterns can be stored in a new repository accessible from the context manager. Offline context prediction would require incorporating into the context prediction model, an additional steps such logging of the tasks sequences into the repository.

Acknowledgments. This research is supported by the ubiquitous computing and network (UCN) project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea. The authors would also like to thank Gonzalo Huerta Canepa for his comments on the draft.

8. References

[1] A. Dey, S. Salber, M. Futakawa, G. Abowd. "An Architecture to Support Context-Aware Applications". *GVU Technical Report GIT-GVU-99-23, 1999.*

[2] A. Dey. "Understanding and Using Context". *Personal and Ubiquitous Computing*, Vol 5, No 1, pp 4-7, 2001.

[3] Baldauf M., Dustdar S., Rosenberg F. "A Survey on Context-Aware Systems". *International Journal of Ad Hoc and Ubiquitous Computing,* 2004.

[4] Biegel G, Cahill V. "A Framework for Developing Mobile, Context-Aware Applications". Trinity College Dublin. *In Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communication,* 2004.

[5] Bradbury, J.; Cordy, J.; Dingel, J.; Wemelinger, M. "A Survey of Self Management in Dynamic Software Architecture Specifications". *Proc. Of the International Workshop on Self-Managed Systems (WOSS'04)*, Newport Beach, California, USA, October/November 2004.

[6] Chen, H. "An Intelligent Broker Architecture for Pervasive Context-Aware Systems". PhD thesis, University of Maryland, Baltimore County, 2004.

[7] Fahy P; Clarke, S. "CASS a Middleware for Mobile Context-Aware Applications". *In Workshop on Context Awareness, MobiSys*, 2004.

[8] Gaia Project. University of Illinois at Urbana-Champaign. At gaia.cs.uiuc.edu

[9] Gu, T; Pung, H; Zhang, D. "A Middleware for Building Context-Aware Mobile Services". *In Proceedings of IEEE Vehicular Technology Conference (VTC 2004)*, Milan, Italy, 2004.

[10] Hnetynka, P.; Plasil, F. "Dynamic Reconfiguration and Access to Services in Hierarchical Component Models". Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Czech Republic. *Proceedings of CBSE*, 2006.

[11] Moran T, Dourish P. "Introduction to Context-Aware Computing". IBM Almaden Research Center, University of California, Irvine. *Special Issue of Human Computer Interaction*. Volume 16, 2001.

[12] R Mayrhofer, H Radi, A Ferscha . "Recognizing and Predicting Context by Learning from User Behavior". Institute for Pervasive Computing, Johannes Kepler Universitat Linz; *Proceedings of The International Conference On Advances in Mobile Multimedia*, Volume 171; September 2003.

[13] Satyanarayanan, M. "Pervasive computing: vision and challenges". Carnegie Mellon Univ., Pittsburgh, PA; *Personal Communications, IEEE,* Volume 8; Aug 2001.

[14] Schilit B, Adams N, Want R. "Context-Aware Computing Applications". *Procsof IEEE Workshop on Mobile Computing Systems and Applications.* 1994. Pages 85-90.

[15] Shang-Wen Cheng; Garlan, D; Schmerl, Sousa J; Spitznagel, B; Steenkiste, P; Ningning Hu. "Software Architecture-Based Adaptation for Pervasive Systems". *Trends in Network and Pervasive Computing - ARCS 2002 : International Conference on Architecture of Computing Systems*, Karlsruhe, Germany, April 8-12, 2002. Proceedings

[16] Tae Seung Ha; Ji Hong Jung; Sung Yong Oh. "Method to Analyze User Behavior in Home Environment". Graduate School of Techno Design, Interaction Design Lab; *Personal and Ubiquitous Computing,* Springer-Verlag London, Volume 10, numbers 2-3; April 2006.

[17] Yu-Sik Park; In-Young Ko; Sooyong Park. "A Task-based Approach to Generate Optimal Software-Architecture for Intelligent Service Robots". Information and Communications University, Sogang University, Republic of Korea, 2007.

[18] Z. Wang and D. Garlan. Task-Driven Computing. *Technical Report CMU-CS-00-154*, School of Computer Science, Carnegie Mellon University, May 2000.

[19] Jini Network Technology.
http://www.jini.org/wiki/Main_Page

[20] Group-aware Ubiquitous Computing Middleware System. Information and Communications University.
http://as.icu.ac.kr/index.html